가상머신을 이용한 실시간 분산처리 시뮬레이터 및 제어기

Development of Real-Time Distributed Simulator and Controller Based on Virtual Machine

양 광 웅, 박 재 현
(Kwang Woong Yang, and Jae Hyun Park)

Abstract : Advanced digital computer technology enables the computer-based controllers to replace the traditional analog controllers used in factory automations. This replacement, however, brings up the side effects caused by the quantization error and non-real-time execution of control software. This paper describes the structure of real-time simulator and controller that can be used for design and verification of real-time digital controllers. The virtual machine concept adopted by the proposed real-time simulator makes the proposed simulator be independent from the specific hardware platforms. The proposed system can also be used in the loosely coupled distributed environments connected through local area network using real-time message passing algorithm and virtual data table based on the shared memory mechanism.

Keywords : real-time simulator, real-time control, virtual machine, object oriented programming

I. 서론

본 연구에서는 네트웍으로 연결된 대규모 플랜트/제어기의 실시간 동작 성능을 분석하기 위한 실시간 분산처리 시뮬레이터의 구조를 제안하고 원형시스템(prototype system)의 구현을 통한 실험 결과를 제시하고자 한다. 컴퓨터 기술의 비약적인 발전으로 대부분의 아날로그 제어기들은 디지털 제어기로 대체되고 있으며, 전기적인 회로로 구성되었던 제어 알고리즘은 제어기 상에서 동작하는 소프트웨어로 바뀌었다. 이와 더불어 제어해야 할 시스템의 범위가 커지고 복잡하게 발전함으로 인해 수학적인 접근이 용이하지 않은, 많은 문제들은 컴퓨터 시뮬레이션을 통해서 시스템의 특성을 해석하고 개선해 나가게 된다. 시스템 해석을 위하여 다양한 패키지가 개발되어 있으나, 현재 주로 사용되고 있는 시뮬레이션 패키지들은 제어 알고리즘의 최적의 해를 구하거나 제어모델의 응답특성을 해석하는데 주 목적이 있는 반면, 실시간 시뮬레이션에는 큰 비중을 두지 않았다. 하지만 설계될 대상 제어기가 발전소 등과 같은 분산 제어기에서 구현될 경우, 실제 구현시 발생할 수 있는 통신망의 지연시간 등에 기인한 제어기의 성능 변화 등은 기존의 수학적 모델링과 시뮬레이션에 중점을 둔 패키지만으로는 완전하게 분석하는데 어려운 점이 많다. 이와 같이 실시간으로 동작하는 플랜트와 연계하여 제어기의 구조를 모델링 하거나, 제어기의 실시간 동작 특성을 확인하기 위해서는 시간 제약조건을 만족하는 분산처리 시뮬레이터를 필요로 한다[1][2]. 본 연구에서는 네트웍으로 연결된 대규모 플랜트/제어기의 실시간 동작 성능을 분석하기 위한 실시간 분산처리 시뮬레이터의 구조를 제안하고 원형시스템 (prototype system)의 구현을 통한 실험 결과를 제시하고자 한다.

실시간 객체 지향형 제어기를 설계하는 방법으로 여러 가지 모델이 연구되어 왔다[3]-[5]. 이들 중 Carnegie Mellon University의 Stewart 등은 포트객체(port-based object)를 이용한 실시간 처리 소프트웨어의 구조를 제안하였다[6]. 하지만 각각의 태스크간에 데이터를 전달하기 위해 전역 상태변수 테이블(global state variable table)을 사용하기 때문에 프로세서간에 비동기적으로 발생하는 메시지의 전달에는 어려운 점이 있으며, 하드웨어와 인터페이스를 하기위해서 포트객체를 이용하는 것은 다양한 종류의 하드웨어 인터페이스 상에서 동작하는 포트객체를 필요로 하여 소프트웨어의 확장성을 떨어뜨린다. Hoger 등은 분산 시뮬레이션(conservative distributed simulation)과 공동 시뮬레이션(concurrent simulation)을 통합한 시뮬레이터 모델을 제안하여 시뮬레이션 모델의 확장성을 높였으나, 노드에 부하가 고르게 분포하지 않을 때 시뮬레이터 성능저하를 보여주고 있다[7]. National Institute of Standards and Technology(NIST)에서는 실시간 제어 시스템(real-time control system; RCS)의 참조모델 구조를 제시하였는데 프로그래머가 실시간 시스템을 프로그래밍 하기위해 필요로 하는 기본적인 API (application programming interface)를 제공하고 있다. 하지만 이 모델은 컴파일 과정에서 플랫폼에 종속적인 코드를 생성하여 실행코드의 호환성을 떨어뜨린다는 단점이 존재한다. 따라서 기존의 연구 결과의 단점을 보완하기 위하여 이기종간의 호환성을 높이고, 실시간 분산처리 시스템의 성능을 시뮬레이션 할 수 있는 새로운 구조의 시뮬레이션 모델의 필요성이 대두되었다. 본 논문에

접수일자 : 1997. 11. 12., 수정완료 : 1998. 10. 9.
양광웅 : 두산기계
박재현 : 인하대학교 항공자동화공학부
※ 본 논문은 96년도 인하대학교 교내연구비에서 지원하여 연구하였습니다.

서는 앞서 언급한 기존의 시뮬레이션 도구의 단점을 보안하고, 실시간 분산처리 환경에서의 모델링, 시뮬레이션은 물론, 나아가 분산처리 제어기의 구현이 가능한 새로운 형태의 시뮬레이션 패키지의 구조를 제안하고자 한다.

본 논문의 구성은 제안하고자 하는 객체지향 시뮬레이터의 특징을 우선 제시하고, 이를 실현하기 위한 가상머신의 구조에 대하여 상세하게 기술한다. 그리고 구현된 실시간 분산 시뮬레이터의 원형 시스템을 이용하여 PID 제어기를 설계하는 예제를 보인다.

II. 객체지향 시뮬레이터

본 논문에서 제시하고자 하는 시뮬레이터의 가장 큰 특징은 포트 객체를 이용한 객체지향형 모델링 기법을 도입한 것과 이를 실시간 분산처리 환경에서 시뮬레이션 할 수 있다는 점이다. 본 절에서는 이들 두 가지 특징에 대하여 설명한다.

1. 객체지향 모델링

현재까지 널리 사용되고 있는 시뮬레이션 패키지들은 모듈화된 모델링 도구를 이용한 계층적인 모델링 방법을 제공하며 이를 토대로 대규모 시스템을 모델링 할 수 있다. 그러나 비록 기존의 패키지들이 모듈화 된 모델링 방법을 허용하고 있으나, 각각의 모듈은 기존의 순차적 프로그램 방법을 고수하고 있기 때문에 궁극적으로는 객체지향 프로그램 기법이 가져다주는 잇점을 살릴 수 없는 실정이다. 예를들어 현재 시뮬레이션 패키지들 중에서 여러 분야에서 가장 널리 사용되는 패키지 중 하나인 MATLAB은 행렬(Matrix)을 기본적인 자료의 단위로 처리하고, M-파일(macro file)이라는 스크립트(script)를 사용하여 하나의 함수를 하나의 M-파일로 구현한다. Matlab을 이용해서 시스템을 시뮬레이션 하거나 분석하는 프로그램을 작성한다면 위와 같은 특성으로 인해 함수간의 자료 공유는 전역 변수를 통해서 이루어지고 프로그램 코드에는 전역 변수를 직접 접근하는 부분이 포함되어진다. 비록 Matlab과 더불어 제공되는 시뮬레이션 도구인 Simulink를 이용하여 시뮬레이션 대상 시스템을 구성하는 각각의 모듈들이 함수별로 나누어져 잘 모델링 되었다고 하더라도 개개 모듈을 구성하는 함수들은 전역 변수로 긴밀하게 결합되어 있으므로, 이미 작성된 모듈(함수)을 다른 시스템의 모델링 및 시뮬레이션에 그대로 이용한다는 것은 쉽지 않다.

이와 같은 순차적 프로그래밍 기법의 단점을 보완하는 모델링 기법으로 객체지향 프로그래밍(OOP: Object-Oriented Programming) 방법이 제시되었다. 모델링 및 시뮬레이션에 있어서 객체지향형 접근 방법에 따라 시스템을 각각의 기능별로 모듈화하고 이를 바탕으로 프로그램을 작성한 후, 각 모듈별로 만들어진 프로그램을 조립하는 식으로 프로그램을 개발하면 모듈간의 결합도를 현저하게 낮출 수 있으며 시스템에 새로운 기능을 추가할 때마다 이미 개발된 프로그램의 전반적인 수정 없이 관련 모듈만의 수정으로도 프로그램을 재사용할 수 있다. 따라서 본 논문에서 제시하는 모델링 기법은 기본적으로 객체지향형 접근방법을 따르며 이렇게 모델링된 시스템은 그대로 객체지향형 프로그램 기법을 이용하여 시뮬레이션 된다.

본 논문에서 제시하는 객체지향형 모델링 방법의 기본 요소는 Carnegie Mellon University의 Stewart등이 실시간 처리 소프트웨어 구조에서 제안한 포트객체(port- based object)이다. 포트객체는 그림 1과 같이 입출력 포트(input, output port)와 객체(object)로 구성되며 객체지향 언어의 캡슐화와 상속성, 내부 정보 숨김 등의 기능이 서로 일치한다. 그러므로 객체지향 언어를 시각적으로 표현하는 도구로 적합한 구조를 가진다. 시각화 된 모델을 구성하는 포트객체는 알고리즘 구성의 최소 단위가 되며 포트객체가 서로 연결되어 하나의 시뮬레이션 모델을 형성한다. 객체의 내부는 외부로부터 숨겨지고 다른 객체와 통신하기 위해 다수의 입출력 포트를 사용한다. 태스크간의 통신을 위해서 포트객체의 입출력 포트는 링크객체(link object)를 거쳐서 다른 태스크를 구성하는 포트 객체와 통신 할 수 있다. 링크객체를 통해 한 개의 출력 포트가 여러 개의 입력 포트에 연결될 수 있다.

그림 2는 다수의 포트객체가 모여서 하나의 컴바인드 포트객체(combined port-based object)로 구성되는 것을 보여주고 있다. 컴바인드 포트객체를 이용하면 모델을 계층구조로 모델링할 수 있다. 컴바인드 포트객체의 입출력 포트는 내부를 구성하는 포트객체의 입출력 포트를 외부의 다른 포트객체들과 연결시켜 준다. 컴바인드 포트객체는 다음과 같은 이점을 제공한다.

① 복잡한 알고리즘을 표현하는 객체들을 작고 단순한 하나의 객체로 묶는 것이 가능하다.

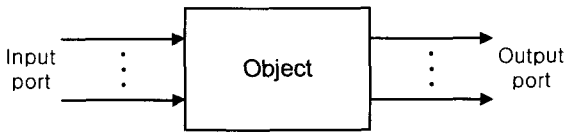
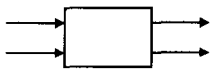


그림 1. 포트 객체.
Fig. 1. Port-based object.



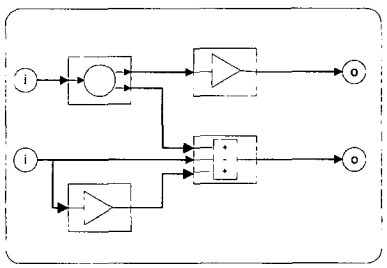


그림 2. 컴바인드 포트객체.
Fig. 2. Combined port-based object.

② 객체들은 객체지향 언어의 캡슐화와 상속성을 이용하여 쉽게 객체지향 소스코드로 바뀐다.

포트객체를 이용하여 모델링된 시스템은 기존의 시뮬레이션 패키지에서 순차적 언어로 번역되는 것과는 달리 객체지향형 시뮬레이션 언어로 기술된다. 즉 포트객체들을 이용해 시각적으로 디자인한 시스템 모델은 전처리 과정을 거쳐 객체지향형 프로그램 소스코드로 변환된다. 자동으로 생성된 소스코드는 컴파일 과정을 통하여 가상머신에서 수행되는 기계어로 번역된다. 이 과정에서 소스코드 내부의 문법 검사, 코드 최적화와 같은 작업을 한다. 특히 기존의 C++/Java 등의 객체지향형 언어와는 달리 제어 시스템을 효과적으로 기술하고 시뮬레이션 하기 위하여 행렬형 데이터를 효과적으로 처리하도록 설계되었다.

2. 가상머신을 이용한 이식성

본 논문에서 제시하고 있는 시뮬레이터의 특징 중 하나는 이기종 분산 시스템에 의한 실시간 시뮬레이션 기능이다. 이는 네트웍으로 연결된 대규모 시스템의 성능을 사전에 검토할 수 있는 필수적인 기능이며 기존의 단일 컴퓨터에서는 구현될 수 없는 기능이다. 네트웍으로 연결된 이기종의 컴퓨터 환경에서 동일한 시뮬레이션 결과를 얻기 위해서는 시뮬레이션 코드의 이식성이 반드시 보장되어야 한다. 본 연구에서는 가상머신(virtual machine)을 도입하여 이식성을 구현하였다.

시뮬레이터 및 제어기의 가상머신은 컴파일 된 기계어 코드를 수행시키는 추상화 된 컴퓨터이다. '가상'이라는 말이 의미하듯 가상머신은 실제의 하드웨어 플랫폼과 운영체제 상에서 구현된 소프트웨어를 말한다. 그러므로 컴파일된 기계어 코드가 수행되기 위해서는 특정한 플랫폼 상에 가상머신이 구현되어 있어야 한다. 가상머신의 대표적인 예는 JVM(Java virtual machine)으로서 자바로 기술된 프로그램은 JVM을 구현한 임의의 시스템에서 실행될 수 있으므로 특정 하드웨어나 운영체제에 독립적인 공통의 환경을 제공한다. 하지만 제어 모델 해석에 주로 사용되는 행렬 연산을 하기 위해서는 상당히 많은 자바 바이트코드(Java byte-code)가 필요하며, 특정한 시스템에 설치된 가상머신에서 emulation되어 수행되는 자바 바이트코드는 수행속도가 5-10배정도 느려지게 된다. 따라서 본 논문에서 제시하는 시뮬레이터 및 제어기의 가상머신에서는 제어 시스템에 적합한 명령어 체계를 확립하고 사용이 빈번한 기능이나 제어 알고리즘은 가상머신 내부에 내장함수 형태로 제공하여 속도를 향상시킨다.

시뮬레이터 및 제어기의 가상머신은 embedded controller와 같은 기계로 이식되어야 할 경우를 고려하여야 한다. 이때 가상머신의 구조는 단순하고 크기가 작아야 하며 이식이 쉬워야 한다. 본 논문에서는 스택 구조를 가진 가상머신을 제안하였다. 스택을 중심으로 한 명령어 처리는, 명령어를 수행할 때 스택으로부터 인수를 가져오고 그 결과를 다시 스택에 되돌린다. 이러한 구조는 내부 레지스터 수가 적게 필요하고 명령어 집합(instruction set)의 구성을 단순화시켜 가상머신을 작고 단순하게 구현할 수 있다는 장점이 있다.

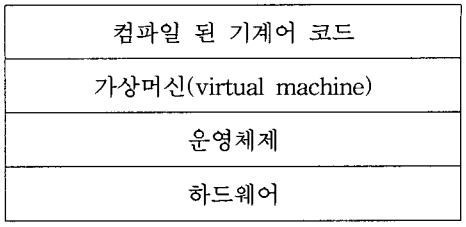


그림 3. 가상머신의 계층구조.
Fig. 3. Virtual machine hierarchy.

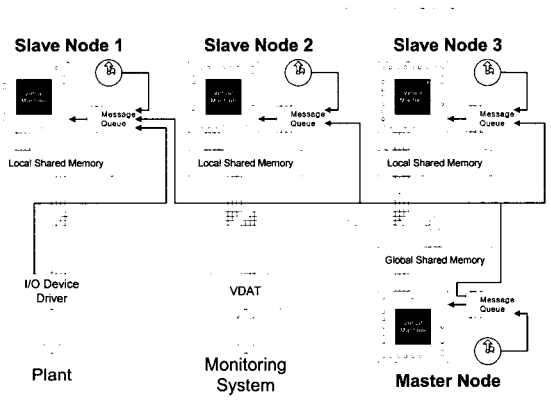


그림 4. 가상머신의 인터페이스.
Fig. 4. Virtual machine interface.

그림 3과 같이 가상머신은 특정한 운영체제와 기계어 코드 사이에서 추상화된 레이어를 제공한다. 그래서 소스코드는 하드웨어 플랫폼이나 운영체제에 무관하게 가상머신에서만 동작하도록 컴파일 된다. 그러므로 가상머신용으로 개발된 프로그램은 하드웨어 플랫폼이나 운영체제가 변하더라도 소스코드를 수정하거나 다시 컴파일해야 하는 부담을 줄일 수 있다. 즉, 가상머신은 시뮬레이터 및 제어기 언어가 이식성을 가지도록 하는데 중심적인 역할을 한다.

가상머신이 다른 노드의 가상머신과 통신하기 위해서 그림 4와 같은 가상머신 인터페이스(virtual machine interface)를 사용한다. 가상머신 인터페이스는 다른 노드의 가상머신과 통신하기 위해서 메시지 큐(message queue)와 공유변수 테이블(shared variable table)을 가진다. 메시지 큐와 공유변수 테이블을 이용한 노드간 통신 구조는 논문 [7]과 [10]에서 제시된 방안을 분산처리 시뮬레이터 및 제어기에 맞도록 수정하여 사용하였다.

분산 환경의 시뮬레이터에서 노드 간 데이터와 메시지 교환 구조는 공동 시뮬레이션에서 프로세서간에 공유되어 사용되는 공유 메모리와, 분산 시뮬레이션에서의 메시지 큐 구조를 혼합하여 사용하였다. 노드에서 발생하는 연속적인 데이터의 전달에는 공유 메모리가 사용되고, 비주기적으로 발생하는 메시지의 처리에는 우선 순위를 가지는 메시지 큐가 사용되었다. 위와 같은 구조는

시뮬레이션 모델의 확장이 용이하고, 각각의 노드에서 수행되는 작업의 동적 스케줄링이 가능하므로 한 노드에 작업이 집중되는 병목현상의 발생을 줄일 수 있다[11][12].

3. 실시간 분산 시뮬레이터

발전소와 같은 대규모 시스템은 다수개의 모듈들이 컴바인드 모듈 형태로 결합되어 모델링 되며, 이러한 시스템은 공간적인 위치나 기능에 따라 나누어져 다중 프로세서에 의해서 처리된다. 본 논문에서 제안한 시뮬레이터는 이러한 환경에서의 동기화된 시뮬레이션을 효과적으로 수행하기 위하여 각 분산 노드의 시뮬레이션 수행 주기를 조절하고 동기화 시키는 메시지를 이용한다. 이러한 구조는 각 노드에 일정한 기저부하로 작용되기 때문에 시뮬레이션이 무리 없이 수행되기 위해서는 일정한 주기로 수행되는 태스크의 평균 수행시간이 메시지의 발생주기보다 짧아야 한다. [7]에서 제시된 분산 시뮬레이션 구조는 한 노드의 태스크가 수행되기 위해서 다른 노드로부터 메시지를 받아서 수행되므로, 메시지가 네트웍을 통해 전송되고 태스크의 실행이 끝난 후 응답 메시지를 네트웍을 통해 다시 전송하게되어 네트웍 자원을 낭비하고 태스크 수행시간도 길어진다. 하지만 각 노드에 클럭을 추가하여 각 노드에 있는 지역 클럭에서 일정한 주기로 메시지를 발생하여 태스크를 주기적으로 수행시키면, 마스터 노드의 메시지에 따라 슬레이브 노드가 태스크를 수행시키는 것 보다 시간을 단축할 수 있으며 네트웍 자원을 적게 소모한다.

메시지 패킷(message packet)은 lifetime, handle, flag, command, parameter로 구성되어 있다. Handle은 메시지를 받을 노드의 주소를 담고 있다. Handle이 0을 가지면 마스트 노드에 대한 handle을 의미하며, 0 이외의 값은 마스트 노드로 전달된 다음 다시 목적 노드로 전달된다. 목적 노드의 메시지 큐는 새로운 메시지를 받아서 스케줄링 알고리즘에 따라 동적으로 메시지의 우선순위를 재배열 시킨다. 스케줄링 알고리즘은 메시지의 잔여수명(residual life-time)이 빠를수록 메시지 큐의 앞쪽에 위치하게 된다. Flag는 긴급, 응답요구 등의 메시지 상태를 기록하는 데 사용된다. 메시지 큐에서 꺼내진 메시지는 command와 parameter에 따라 태스크에서 수행되는 작업의 종류가 결정된다.

주기적인 메시지는 주로 입출력 디바이스로부터 데이터 수집, 모니터링 시스템으로 데이터 출력, 제어 알고리즘의 반복 수행 등의 작업을 통하여 처리되는 연속성을 가지는 데이터 교환을 목적으로 가상머신의 인터페이스를 사용한다. 기존에는 다양한 물리적 전송매체에 대한 각각의 디바이스 드라이버를 제어하여 데이터를 주고 받았지만, 사용자 프로그램과 디바이스 드라이버 사이에 추상화된 네트웍 레이어를 둠으로 사용자는 노드간에 연결된 물리적 매체에 무관하고 일관되게 데이터를 주고받는 프로그램을 작성하는 것이 가능하다. 인터페이스의 지역 공유메모리(local shared memory)와 전역 공유메모리(global shared memory)는 공유메모리(shared memory)와 같은 역할을 하도록 Ethernet, field-bus, 비동기 통신과 같은 네트웍 레이어를 추상화시킨 상위 레이어로서, 가상머신 인터페이스에 의해서 주기적으로 데이터의 동기를 유지시킨다.

모든 노드는 지역 공유메모리를 가지고 태스크가 수행될 때 필요한 전역변수들을 저장하고 있다. 마스터 노드는 슬레이브 노드와 달리 전역 공유메모리를 가진다. 전역 공유메모리는 여러 개의 노드로부터 접근되어지지만 하나의 노드만 엑세스 할 수 있기 때문에 테이블의 lock/unlock 하는 과정이 필요하며, 지역 공유메모리와 전역 공유메모리는 네트웍으로 연결되기 때문에 네트웍의 성능에 따라 접근시간을 많이 소비하게 된다. 그러므로 슬레이브 노드에서 수행되는 태스크는 직접적으로 전역 공유메모리의 변수를 접근하지 못하고 지역 공유메모리를 통해서 한 주기의 마지막 사이클에서 한번 접근 하여 지역 공유메모리의 내용을 전역 공유메모리의 내용과 동기시킨다. 그러므로 최소한의 오버헤드(overhead)로 수행되는 것이 가능하다.

에러 메시지나 경고 메시지와 같이 비주기적으로 발생하는 이벤트(event)는 타임 스탬프를 가지는 메시지 형식으로 노드에 전달되어 메시지 큐에 삽입된다. 이와 같은 비주기적으로 발생하는 메시지들 중에는 중요도가 높은 것이 많으므로 대부분 가장 높은 우선순위를 가지고 메시지 큐에서 스케줄링 되어 메시지 큐의 앞부분에 삽입된다.

III. 분산 시뮬레이션을 위한 가상머신의 구조

본 절에서는 제안된 시뮬레이터의 내부 구조를 가상머신을 중심으로 설명한다. 가상머신의 내부 구조는 그림 5에서 보여주고 있다. 가상머신은 크게 주처리장치(CPU), 메모리(memory), 내장 코드(internal code), 입출력부(input/ output)로 구성되어 있다. CPU는 레지스터(register), 명령어 패치(instruction fetch), 그리고 수행부(executor)로 구성되어 있다. 메모리에는 코드(code), 데이터(data), 스택(stack), 힙(heap) 영역으로 구분되어 있다. 내장 코드는 연산자(operator)와 내부함수(internal function)로 구분된다. 입출력부를 통해 가상머신이 외부와 연결된다.

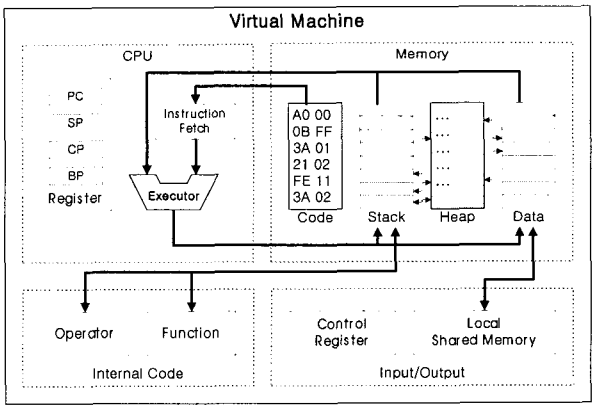


그림 5. 가상머신의 내부구조.

Fig. 5. Internal structure of virtual machine.

1. 주처리 장치

주 처리장치 부분은 코드 메모리로부터 명령어를 읽어와 해석하고 수행시키는 역할을 한다. 그림 5와 같이 가상머신은 스택을 중심으로 동작 하기 때문에 표 1처럼 PC, SP, BP, CP, AU, BU, C, D 등의 적은 수의 레지스터(register)를 가진다. PC, SP, BP, CP 레지스터는 메모리상의 수행되고 있는 코드의 위치나 참조되는 데이터의 위치를 가리키는 포인터 레지스터로 사용된다. AU, BU, C, D 레지스터는 데이터 저장을 위해 사용된다.

명령어는 8-bit의 op-code와 32-bit operand로 구성되어 있다. 코드 메모리 영역의 명령어는 명령어 패치를 거쳐 수행기로 전달되어 의미가 해석되고 실행된다.

수행기는 PC 레지스터가 가리키는 곳의 코드 메모리 영역에서 명령어를 가져와 해석하고 이를 수행한다. 연산자(operator)는 논리연산과 사칙연산을 제공하며, 함수(function)에서는 행렬이나 백터에 관련된 함수들을 제공한다. 가상머신은 호환성에서는 뛰어난 반면, 인터프리터이기 때문에 속도가 느리다는 단점이 있다. 하지만 알고리즘이 복잡하고 수행시간이 느린 기능들은 태스크의 수행속도를 최대한 증가 시키기 위해서 가상머신과 같이 컴파일 되어 하드웨어 플렛폼의 CPU 명령어 코드를 그대로 사용하게 된다.

2. 메모리

C/C++에서 메모리를 읽고 쓰기 위해 사용하는 포인터의 개념은 디버깅이 어려우며 프로그램 실행중 예기치 않은 오류를 많이 발생시킨다. 시뮬레이터 및 제어기의 가상머신에서는 포인터 처리를 없애고 행렬 데이터형을 도입함으로 실행 중 발생할 수 있는 오류를 근원적으로 줄일 수 있으며, 메모리 포인터의 사용으로 인한 주소 기억의 필요성을 없앴다. 가상머신에서 사용되는 메모리는 코드 메모리(code memory) 영역, 데이터 메모리(data memory) 영역, 스택 메모리(stack memory) 영역, 그리고 힙 메모리(heap memory) 영역으로 구성된다. 코드 영역에는 가상머신에서 수행될 기계어 코드가 저장되며, 데이터 영역에는 전역변수의 저장공간으로 사용되고, 함수의 호출이나 함수 내부의 지역변수 저장에도 스택 영역이 사용된다. 힙 영역에는 프로그램 수행 중 가변적인 크기를 가지는 변수가 사용하는 메모리 공간이 할당된다.

3. 내장코드(internal code)

사용자가 알고리즘을 구현할 때 기본적으로 필요로 하는 함수들은 연산자(operator), 내부함수(internal function), 그리고 라이브러리 함수(library function)의 세 가지 형태로 구성된다. 연산자란 기본적인 산술 연산을 하는 함수들을 말하며 기호 형태를 가진다. 내부함수는 연산자와 같이 가상머신의 내부에서 구현되어 있지만 호출 방식이 함수와 같다. 라이브러리 함수는 소스코드나 목적코드 형태로 제공되며 컴파일 과정을 거쳐 가상머신용 코드로 만들어진 함수를 말한다. 가상머신에서는 실수와 복소수의 기본적인 사칙 연산자와 수학 함수를 제공하며, 행렬이나 다항식에 관한 연산과 조작을 위한 다양한 연산자와 함수들을 제공한다. 또 자료를 문자단위로 입출력하거나 그래프 윈도우에 그래프를 출력하거나 그래프를 조작해주는 함수들이 있다. 연산자와 내부 함수는 가상머신이 디자인 될 때 같이 만들어진 함수들이므로 라이브러리 함수나 사용자가 작성한 함수보다 속도 면에서 빠르다는 장점을 제공한다.

IV. 시뮬레이터의 구현 및 실험

본 논문에서 제안된 전체 시스템은 포트객체 편집기, 전처리기, 컴파일러, 가상머신, 가상머신 인터페이스로 구성되어 있으며 이들 중 포트객체 편집기를 제외한 모든 부분을 실제로 구현하여 시험하였다.

제어기 및 시뮬레이터는 제어 알고리즘을 수행시키기 위해서 내부에 가상머신을 포함하고 있다. 가상머신은 제어기 및 시뮬레이터와 직접 연결되지 않고 가상머신 인터페이스를 거쳐서 연결된다. 가상머신 인터페이스는 내부에 메시지 큐와 공유메모리를 가지고 있어서 가상머신이 분산환경에서 실시간으로 메시지와 데이터를 주고받으며 동작하도록 한다.

제어기 및 시뮬레이터와 가상머신은 C++ 언어로 개발되었으며, 현재 소스코드는 Intel사의 386, 486, Pentium CPU를 사용하는 PC 기반의 플렛폼과 Windows 95/NT 운영체제에서 동작하도록 컴파일 되었다. 제어기 및 시뮬레이터는 15,000 라인 분량의 C++언어로 개발되었다. 가상머신 인터페이스는 10,000라인 분량의 C++언어로 개발되었으며, 라이브러리 형태로 컴파일 되어서 제어기 및 시뮬레이터 프로젝트에서 링크된다. 가상머신은 약 30,000 라인 정도의 C++언어로 구현되었으며 Windows 동적 링크 라이브러리(Dynamic Link Library; DLL)로 구현되어 있어서 후에 기능을 확장하는 것이 용이하다. 가상머신 중 20,000 라인정도가 내장함수를 위한 분량이다. 또한 시뮬레이터와 제어기가 다기종으로 구성된 컴퓨팅 환경에서 동작하기 위해서는 가상머신이 다양한 하드웨어 플렛폼과 운영체제에 따라 개발되어야 한다.

실시간 분산처리 시뮬레이터가 플랜트 시뮬레이션에 적용되는 구조는 온라인 시뮬레이션(on-line simulation)과 오프라인 시뮬레이션(off-line simulation)의 두 가지로 나누어진다. 온라인 시뮬레이션에서는 제어기에 실제의 플랜트가 연결되어 시뮬레이션이 이루어진다. 제어기의 제어 알고리즘(task)은 정해진 시간 내에 수행을 마쳐서 다음 제어값을 플랜트에 넘겨주어야 하므로 제어 알고리즘의 수행시간이 실시간 제약조건을 만족하는지를 시뮬레이션으로 확인할 수 있다. 오프라인 시뮬레이션은 모델링된 플랜트가 분산시스템의 노드로 대체되어 플랜트의 역할을 한다. 하지만 제어기는 플랜트와 통신하기 위해서 공유메모리를 통해 변수를 입출력 할 뿐 플랜트가 실제이건 모델링된 소프트웨어 알고리즘이건 고려할 바가 아니다. 그러므로 오프라인으로 시뮬레이션한 모델을 가상의 공정만 실제로 대체시키고 제어기 부분은 수정하지 않고도 온라인 제어에 사용할 수 있다.

본 연구에서 설계되고 구현된 실시간 분산처리 시뮬레이터 및 제어기의 효용성을 보이기 위하여 분산환경에

서 실시간으로 운용되는 시뮬레이터를 이용하여 PID 제어기를 설계해 본다. 아래의 예제는 제어 대상체가 비교적 느리게 반응하는 시스템으로 공정 제어와 같은 시스템이다. 제어 대상의 시스템 모델은

$$G(s)=\frac{5}{s^2+6s+5}$$

이다. 이 모델은 그림 8과 같이 아무런 제어를 하지 않더라도 그 자체로서 이미 안정한(stable) 시스템이다. 하지만 시스템의 동작 특성은 원하는 동작을 하는데까지 걸리는 시간이 긴 느린 시스템이다. 이 시스템을 비교적 반응 시간이 빠르도록 PID 제어기를 설계해 본다. PID 제어기의 계수를 결정하는 방법은 Ziegler-Nichols의 PID 계수조정, 극 배치(Pole Placement)를 이용한 PID 계수 조정법 등이 주로 사용되고 있다. 하지만 여기서는 사용자가 원하는 결과를 얻을 때까지 시뮬레이션을 반복하여 계수를 조정하도록 하였다.

제어기와 공정의 블록 다이어그램은 제어기, 공정, 감시기, 3개의 노드로 나누어 진다. 전체적인 구성은 그림 6과 같다. 감시기는 제어 신호를 만들어내고 공정으로부터의 출력을 감시한다. 제어기는 PID 제어 알고리즘으로 구성되어 있다. 감시기는 마스터 노드 역할을 하며 제어기와 공정은 각각 하나의 슬레이브 노드로 구성된다. 시뮬레이션이 시작되면서 감시기에서 단위계단응답을 제어기와 공정으로 전달한다. 시뮬레이션이 진행되면 공정의 출력을 수집하여 감시기에서 그래프로 보여준다.

그림 6에서 설계된 제어기와 공정 블록 다이어그램이 시뮬레이터에서 실행되기 위해서는 객체지향 소스코드로 바뀐 후 다시 컴파일러에 의해 컴파일 되어야 한다. 그

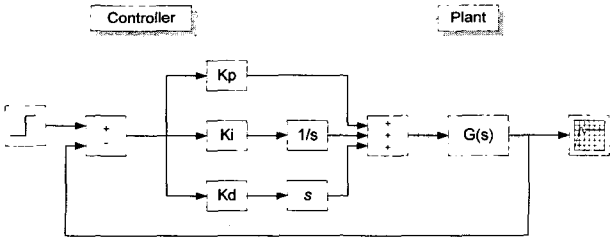


그림 6. 제어기와 공정의 블록다이어그램.
Fig. 6. Block diagram of controller and plant.

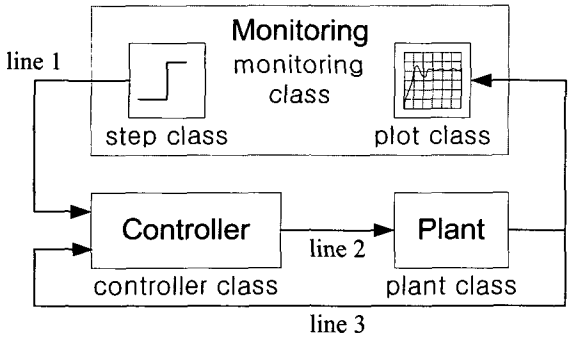


그림 7. 제어기, 공정, 감시 모듈.
Fig. 7. Controller, plant, and monitoring module.

림 7에서와 같이 시스템을 제어기(controller), 공정(plant), 감시기(monitoring)로 나누고 각각 controller class, plant class, monitoring class로 구성한다. simulation class가 노드를 구성하는 class들을 상속하여 선언되고, 시뮬레이터는 simulation class의 main 함수를 호출하여 시뮬레이션을 진행시킨다.

제어하고자 하는 공정이 비교적 느리게 동작하므로 샘플링 시간을 100ms로 하고 10초 동안 시뮬레이션 하였다. 원하는 응답을 얻기 위해서 Kp, Ki, Kd 값을 바꾸어 가며 반복 시뮬레이션 하였다. Kp, Kd를 고정하고 Ki를 0.1에서 3까지 y축으로 변화시켰을 때의 단위계단 응답은 그림 9와 같다. Ki가 작은 값을 가질 때는 1에 서서히 수렴하고 Ki가 1.5보다 커지면 그래프가 진동하는 것을 볼 수 있다. 이러한 과정을 반복하여 PID제어기의 각 계수가 Kp = 1.4, Ki = 1.1, Kd = 0.1 일 때 시뮬레이션 결과는 그림 10과 같다. 2초 부근에서 약간의 과도응답을 보이지만 정상상태에 도달하는 시간이 개루프보다 두 배 이상 빨라진 것을 볼 수 있다.

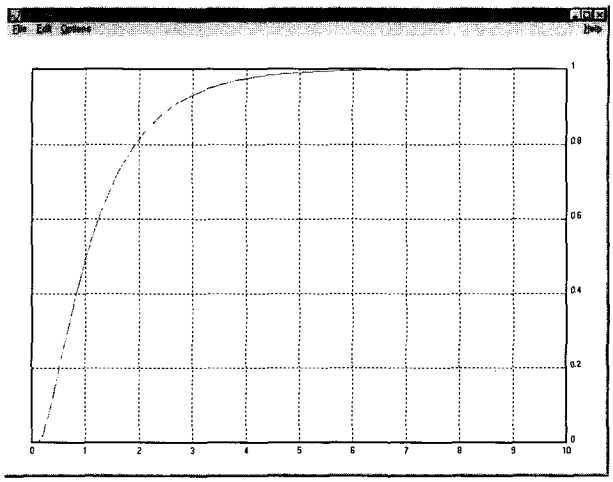


그림 8. 공정의 과도기 응답.
Fig. 8. Transient response of plant.

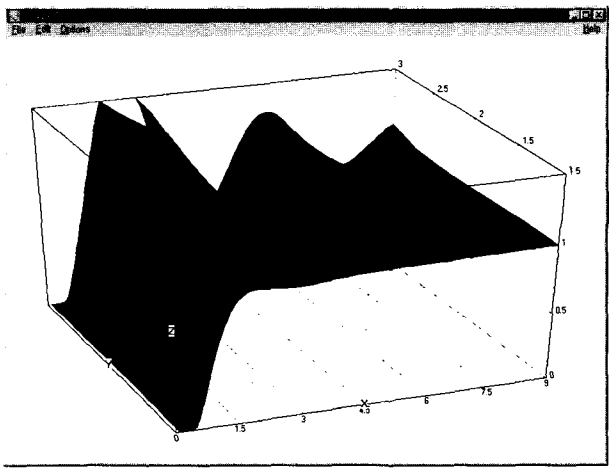


그림 9. PID 제어기의 계수 결정.
Fig. 9. Determination of PID coefficient.

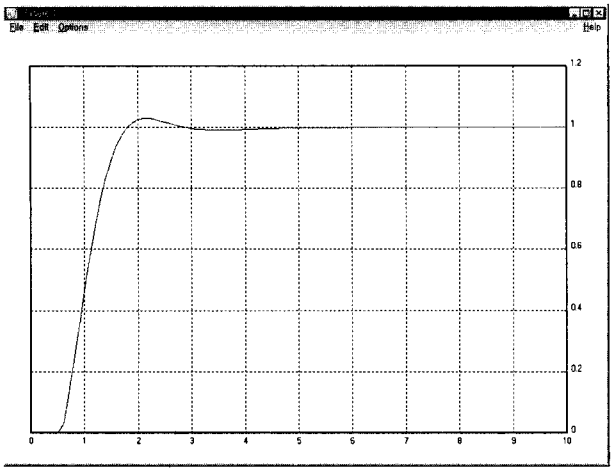


그림 10. PID 제어기와 공정의 과도응답.
Fig. 10. Transient response of controlled plant.

V. 결론

본 논문에서는 실시간 분산처리 시뮬레이터를 위한 객체지향형 시뮬레이터의 기본 구조를 제시하고 이의 구현을 위하여 시뮬레이터의 내부를 구성하는 가상머신의 구조를 제시하였다. 또한 제안된 시스템을 실제로 구현하여 실험함으로써 다음과 같은 성과를 얻을 수 있었다. 첫째, 실시간 분산처리 시뮬레이터 및 제어기에서 수행되는 제어 알고리즘의 개발에 객체지향 프로그래밍 언어를 사용하여 모델을 구성하는 모듈간의 결합도를 낮추고 모듈의 재사용성을 높일 수 있다. 둘째, 제어기 구조에 가상머신을 도입하여 다양한 컴퓨터로 구성된 분산환경에서의 이식성을 제공한다. 그리고 사용이 빈번한 기능이나 제어 알고리즘은 가상머신 내부에 내부함수 형태로 제공하여 속도를 증가시켰다. 셋째, 분산처리 시뮬레이터의 제어 알고리즘은 두 시스템 간의 공통된 인터페이스를 제공하는 메시지 큐 와 공유메모리를 통하여 제어대상과 연결되므로, 제어대상을 실제의 플랜트가 아닌 가상의 플랜트 모델을 설정하여 제어 알고리즘의 성능을 분석할 수 있다. 위에서 제시하는 실시간 분산처리 시뮬레이터 및 제어기 구조는 다 기종으로 구성된 지역 네트웍 환경을 수정 없이 이용 가능하고 제어 알고리즘의 개발에는 객체지향 방법을 사용하므로, 시스템 개발 기간을 단축할 수 있으며 개발된 시스템의 유지보수 비용을 줄일 수 있다.

참고문헌

[1] S. Bennett, *Real-time Computer Control*, Prentice Hall, Maylands Avenue Hemel Hempstead, 1994.
[2] M. A. Piera, C. de Prada, "Modeling tools in the continuous simulation field," vol. 66, no. 3, pp. 179-189, March, 1996.
[3] K. H. Kim, "Toward new-generation object-oriented real-time software and system engineering," *SERI Journal*, vol. 1, no. 1, pp. 1-23, 1997.
[4] M. de Champlain, "A small and expressive object-based real-time programming language," *SIGPLAN Notices*, vol. 25, no. 5, pp. 124-134, 1990.
[5] C. G. Cassandras, *Discrete Event Systems: modeling and performance analysis*. Richard D. Irwin, Inc., and Aksen Associates, Inc., 1993.
[6] D. B. Stewart, R. A. Volpe and P. K. Khosla, "Design of dynamically reconfigurable real-time software using port-based objects," *Technical report, Carnegie Mellon University*, Pittsburgh, PA 15213, July, 1993.
[7] H. R. Hoger and D. W. Jones, "Integrating concurrent and conservative distributed discrete-event simulators," *Simulation*, vol. 66, pp. 303-314, Nov., 1996.
[8] J. Gosling and H. McGilton, "The Java language environment a white paper," *Sun Mic- rosystems Computer Company*, October, 1995.
[9] D. R. Gentner and J. Grudin, "Design models for computer human interfaces," *IEEE Computer*, vol. 29, no. 6, pp. 28-35, June, 1996.
[10] A. Pullafito, S. Riccobene and M. Scarra, "Evaluation of performability parameters in client-server environments," *The Computer Journal*, vol. 39, no. 8, pp. 647-662, 1996.
[11] K. Ramamritham, J. A. Stankovic, "Efficient scheduling algorithms for real-time multiprocessor systems," vol. 1, no. 2, pp. 184-194, April, 1990.
[12] H. D. Karatza, "Simulation study of task scheduling and resequencing in a multiprocessing system," *Simulation*, vol. 68, no. 4, pp. 241-247, April, 1997.

양 광 웅

1996년 인하대 자동화공학과 졸업. 동대학원 석사(1998). 1998년-현재 두산기계 근무. 관심 분야는 객체지향형 프로그램 기법, 공장 자동화, 컴퓨터 응용 제어 시스템, 실시간 제어시스템.

박 재 현

1986년 서울대학교 제어계측과 졸업. 동대학원 석사(1988), 동대학 박사(1994). 1995년-현재 인하대학교 기계항공자동화공학부 조교수. 관심분야는 실시간 컴퓨터시스템 구조, 컴퓨터 통신망, 실시간 자동제어 시스템, 개방형 제어시스템.